N94-23966

SOFTWARE REUSE IN SPACECRAFT PLANNING AND SCHEDULING SYSTEMS

David McLean, Alan Tuchman, Todd Broseghini, Wen Yen, Brenda Page, Jay Johnson, Lynn Bogovich, Chris Burkhardt, James McIntyre, Scott Klein, Hung Dao and Ronald Littlefield

Bendix Field Engineering Corporation (Allied-Signal Aerospace Co.)
Aerospace Building, Suite 500
10210 Greenbelt Road, Seabrook, MD 20706, USA

William Potter

Spacecraft Control Programs Branch
Goddard Space Flight Center
Greenbelt, MD 20771, USA

ABSTRACT

The use of a software toolkit and development methodology that supports software reuse is described. The toolkit includes source-code-level library modules and stand-alone tools which support such tasks as data reformatting and report generation, simple relational database applications, user interfaces, tactical planning, strategic planning and documentation. The current toolkit is written in C and supports applications that run on IBM-PCs under DOS and UNIX-based workstations under OpenLook and Motif. The toolkit is fully integrated for building scheduling systems that reuse AI-knowledge base technology. A typical scheduling scenario and three examples of applications that utilize the reuse toolkit will be briefly described. In addition to the tools themselves, a description of the software evolution and reuse methodology that was used is presented.

Key Words: Scheduling, planning, AI technology, C-based software, software reuse

1. INTRODUCTION

The Spacecraft Control Programs Branch/Code 514 at NASA's Goddard Space Flight Center, with support from Bendix Field Engineering Corporation (BFEC), has been building spacecraft planning and scheduling systems for ground support operations since 1985. Although this technology is Artificial Intelligence (AI) based, it is also C-based because of the requirement for portability and high performance (McLean, et al., [1]).

Three main applications developed to date are: the Earth Radiation Budget Satellite (ERBS)-Tracking and Data Relay Satellite System (TDRSS) Contact Planning System (CPS) (McLean, et al, [2]), the Explorer Platform Planning System (EPPS) (McLean, et al., [3]), and the Hubble Space Telescope Servicing Mission Planning and Replanning Tool (SM/PART) (Johnson, et al., [4]). The ERBS system runs on an IBM AT, while the EPPS and SM/PART systems run on UNIX-based workstations under Open Look and Motif.

The AI technology used in these applications includes the use of a tactical planning tool called the Planning And Resource Reasoning (PARR) shell. PARR is a heuristic based scheduler which uses frame-like data structures to represent scheduling knowledge. PARR differs from most other scheduling shells in that it places a great deal of emphasis on conflict resolution versus conflict avoidance (McLean et. al., [5]).

Before PARR can be used, activity classes must be defined and an external scheduling environmental data timeline containing potential resources and constraints must be available. In addition to PARR, a strategic planning tool is available that allows the user to specify classes of activities in terms of constraints, resources and conflict resolution strategies and then to store these activity classes in a Knowledge Base (KB).

For satellite scheduling, a scheduling environmental data timeline is generated by the Flight Dynamics Facility (FDF) at Goddard. When the activity KB and the environmental data timeline become available, the user can invoke PARR to autonomously generate a conflict-free schedule. Later, the user can invoke an interactive version of PARR to browse and edit the schedule.

The task of building satellite scheduling systems at Goddard has led to the reuse and evolution of earlier versions of the software. This effort has resulted in a development methodology along with new software tools for managing software reuse and evolution.

2. TYPICAL SCHEDULING SCENARIO, REQUIREMENTS AND PROBLEMS

2.1 A Typical Scenario

The scheduling scenario given here is based on satellite scheduling tasks at Goddard. It is a generalization drawn from interviews with several Flight Operations Team (FOT) schedulers.

1. Get external scheduling environmental data from FDF.
2, Define classes of activates to be scheduled.
3. Put external scheduling environmental data on timeline.
4. Put activities on timeline in predefined order.
5. Perform resource allocation, constraint checking and conflict resolution, as required.
6. Generate tentative schedule.
7. Interactively refine schedule until deadline.
8. Expand final schedule into detailed command schedule.
9. Send final schedule to the Command Management System (CMS) for load generation.
10. Keep careful records of what was done.

2.2 Software Requirements

The software requirements for each stage of the scenario include:

1. Establish Ethernet link to the source of external environmental data. Reformat external data to internal scheduling formats.
2. Provide a tool which takes an activity class definition and generates a KB entry for each activity class.
3. Provide a tool which places the external environmental data on a timeline.
4. Provide a tool which places the activities specified by each activity class on the timeline.
5. Perform automatic resource allocation, conflict checking and heuristic conflict resolution.
6. Write the tentative schedule to a file and print a report.
7. Provide graphic timeline browse and edit capabilities.
8. Provide a tool which allows the user to specify sequences of low-level command activities that represent the details of the higher level activities. Provide a tool which expands the higher level activity schedule into the lower level command sequence schedule.
9. Reformat the internal schedule to the external schedule format and establish the Ethernet link to the CMS.
10. Provide a user interface and generate reports so that each of these steps can be easily accomplished and carefully managed.

2.3 Scheduling Problems

At the heart of the scheduling scenario are four basic types of problems that humans encounter in placing spacecraft activities on a schedule:

1. Order of activity placement
2. External Constraints
3. Spacecraft Resources
4. Conflict resolution

When a schedule becomes crowded with many activities it becomes more and more unlikely that new activities will have enough resources available, so that the order of activity placement becomes an important conflict avoidance heuristic. A typical response to this problem is to place the most constrained activities on the timeline first.

Scheduling and planning problems involve predicting external conditions and using the constraint information to decide where activities should be put on a schedule. From the scheduling point of view, external events may be viewed as constraints or resources. As spacecraft activities are placed on a timeline, they are initially constrained by the predicted

orbital events (initial external constraints) and later by other activities already placed on the schedule (current external constraints).

Some types of activities may require the use of sharable or consumable spacecraft resources such as power or a tape recorder. Models of the behavior of these resources are required in order to reflect the changes to the scheduling environment as activities of this type are placed on the timeline.

When conflicts are found due to constraint violations or resource unavailability, heuristics may be used to resolve these conflicts. For example, a conflicting activity may be scheduled at a later time or may use an alternative resource.

The IEPS software toolkit used to solve these problems allows the human scheduler to specify four types of scheduling heuristics corresponding to the four problem types described. A strategic planning tool that allows the user to specify each of these types of heuristics and store them in a the strategic planning KB is provided.

3. EVOLUTION OF THE TOOLKIT

3.1 ERBS System

The purpose of the ERBS scheduling system is to aid the user in building a confirmed TDRSS contact schedule. This is done by specifying the desired times for contacts and sending these requests to the Network Control Center (NCC). Requests for TDRSS time may have to be resubmitted several times when conflicts occur. This iteration with the NCC for confirmed TDRSS time is often a very time consuming task.

The IEPS toolkit started by implementing only the ERBS system requirements, which are a subset of all the requirements in the scenario described above. For example, the Ethernet connection was never available, so a tape drive was provided to obtain the external scheduling environment data file from the FDF. Also, the product of the ERBS scheduling system (a set of reports) had to be hand carried to the mission planning terminal. The ERBS KB building activity was done as a traditional knowledge acquisition task by a software/knowledge engineer. Finally, ERBS did not require the expansion of the activities into more detailed command sequences because that task was done in the CMS.

After delivering the ERBS scheduling system, the IEPS toolkit consisted of:

1. A set of reusable C modules.
2. A reformatter for the external environmental data timeline.
3. A batch/interactive tactical planning shell (PARR).
4. A set of report generators.

During the maintenance phase of the ERBS system, the toolkit was reengineered so that the system was more maintainable. This reengineering effort included a greater separation of the user interface code from the rest of the system code.

Anticipating future work, a copy of the ERBS software was ported to UNIX-based workstations where the user interface software was reengineered to interface with X-Windows and the reformatting and report generation tools were enhanced and made more generic.

3.2 EPPS

When the EPPS project started, the IEPS team was prepared to reuse a good portion of the ERBS system. For EPPS, PARR was broken down into several more specialized scheduling tools: interactive, batch and merge. Also, a two-tape-recorder model was added to PARR's resource modeling capabilities.

An additional requirement for EPPS was the capability to define sequences of activities that represent the details of each higher level (traditional PARR) activity. Also, there was a requirement to expand the PARR schedule file into time-instantiated sequences. The definition capability was accomplished by use of the database tools and the expansion was done by a special stand-alone tool.

A major development effort for EPPS was to build a KB editor. Early versions of the KB editor utilized the IEPS toolkit user interface tools. Later versions of the KB editor also

utilized the IEPS relational database toolkit. EPPS currently uses the Open Look user interface library, but future applications will be using Motif.

EPPS has Ethernet connections to the CMS to receive external environmental data files and science schedules. The final schedule produced by EPPS is sent to the CMS via this Ethernet connection. Because of the variety of different data types available electronically, the IEPS reformatting toolkit was greatly expanded.

A separate power model was developed for EPPS. This is a sophisticated simulator that is utilized by the FOT to keep track of the health and safety status of the batteries. At the current time, this power model is not an integral part of PARR. However, there are plans to merge the power model with the interactive version of PARR so that the FOT can see the effects of interactive additions or deletions of activities on the health and safety status of the batteries. There are plans to build a simulation toolkit which would support all of the (currently) separate spacecraft simulation efforts.

Documentation tools have also evolved, and these tools have saved many hours of time when documentation for deliveries are required. Documentation tools are used extensively to generate maintenance documents.

By the time of Explorer Platform launch (spring 1992), the IEPS toolkit included the following (NASA-GSFC [6]):

1. An extended set of C modules in various libraries.
2. A set of reformatting/report-generation tools.
3. A set of user interface tools.
4. A set of database tools.
5. An activity class definition tool and a command activity definition tool.
6. A set of batch tactical scheduling tools.
7. An interactive tactical scheduling tool.
8. A set of documentation tools.

3.3 SM/PART

The Hubble Space Telescope (HST) in-orbit servicing activities require detailed planning of the preferred and alternative activities. In addition to these activities, "command plans" are also required to describe the detailed commands that are to be sent during the mission.

The HST servicing mission scheduling needs are quite a bit different from the other two missions described. One of the differences is that the HST activities are very specialized--the concept of activity class does not fit most of these uniquely defined activities. Nevertheless, virtually every category of the IEPS toolkit contributed to and was enhanced by the SM/PART effort. For example, the database and KB editor tools were enhanced and the code used to interface with Open Look, used by EPPS, was converted to Motif. Also, new user interface displays were created to include different types of timeline displays, PARR's scheduling algorithm was modified to support the more demanding temporal dependency links between activities and at the module level, the IEPS toolkit was extended to include more date/time reformatters.

4. USING THE TOOLKIT TO BUILD SCHEDULING APPLICATIONS

The first step in building new scheduling applications is to configure the reformatting tools and user interface files to specify the domain of activities in terms of forms and menus. Then, script files (shell scripts) are created to link the various tools into a unified system. This unified system is then iteratively refined by reconfiguring and extending the capabilities of the tools as well as by developing new tools until the system meets all the initial and discovered requirements. Rapid prototyping is often used during the initial stages of building new applications in order to enhance the requirements discovery process.

The software engineer starting a new scheduling application would most likely examine the basic requirements and then select the current application that meets most of these requirements. Those parts of the system scripts and menu options which are not required would be eliminated. In addition, the reformatting tools would be reconfigured so that the external scheduling environmental data could be read by PARR.

Next, the names of the external environmental data, in the activity specification tool's menu file,

would be changed to reflect the new application. Any other required changes to these menu files would also be made at this time, until the user is satisfied that the system can adequately specify the types of activities required.

At this point, if a test external environmental data file is available, the user can start using PARR to generate test schedules from the prototype activity classes defined. During this process, the user learns how to use the system and discovers the types of activities that produce the most desirable schedule. This process may go on for some time, but it will eventually establish the basic activity classes required.

Finally, the output schedule is reformatted according to the requirements of the user, and TCP/IP tools may be used to receive resource files from an external source and send the final schedule to an external user. Various reports which allow the user to manage the entire scheduling task will also be prototyped via the report generation tools.

5. MANAGING SOFTWARE EVOLUTION

5.1 Evolutionary Prototyping versus the Waterfall Model

Evolutionary prototyping is a software development methodology that combines the advantages of software reuse and prototyping. Because evolutionary prototyping is based on software evolution, special care must be taken to provide a management structure which supports it (Arthur, L., [7]). The traditional waterfall model does not accommodate the sort of evolutionary development made possible by rapid prototyping capabilities. Further, the "manufacturing model" of software development makes the unrealistic assumption that software can be totally pre-specified as with a mass-produced item.

On the other hand, the evolutionary prototyping approach makes the assumption that software can never be totally pre-specified because the development process is, by nature, a discovery process which results in a continuously evolving specification. Thus, evolutionary prototyping provides a methodology for requirements discovery through software reuse and prototyping.

Lowel Arthur contends that the term "rapid prototyping" is regarded with suspicion by many customers because many of the traditional prototyping efforts were done in labs away from the potential users with hardware and software that was far removed from the target platform. The result of this type of "throw-away" prototyping effort often led to systems that had little relevance to what the customers could use and were very costly to completely reengineer.

By contrast, evolutionary prototypes are developed with reuse in mind and are built on a stable, reuse library. They are developed in the same language and on platforms similar to the target system. Typically reuse tools are small, well-integrated, and easy to understand and maintain. Reuse tools should also be generic enough to be of use in a wide variety of applications. The rapid, evolutionary prototyping paradigm is a plan-do-check-act cycle which is repeated until all the requirements are met, at which time the system is delivered.

Some of the advantages of evolutionary prototyping are:

1. It is simple to do. Prototyping means concentrating on the essentials of the problem.
2. It provides value in a short time frame with early, easy wins that establish customer commitment.
3. It allows engineers and customers to learn as they go.
4. Often, until something can be demonstrated, the customer will not be able to clearly articulate the needs for the system.
5. It lowers risk because the project can be canceled at any time. When new versions of the prototypes are delivered, managers and customers know the exact state of the project.
6. It increases the ability to deliver bug-free systems with the desired functionality.

A potential management problem with reuse is that traditional productivity measures, such as lines of code, do not reflect the effort and quality of this process. Further, software reuse requires an initial training phase for the software engineers. Therefore, these factors need to be

taken into account when predicting the software effort for the evolutionary prototyping approach.

5.2 IEPS Activities and Software Reuse

The IEPS group consists of software engineers who spend some of their time keeping pace with cutting-edge software technologies. In addition to developing new applications, they look for new technologies that promise to be of value to mission operations systems at Goddard. These activities include:

1. Periodic reviews of new technologies as presented at various software conferences and in journals.
2. Emphasis on software reuse of independent but well integrated tools.
3. Use of rapid prototyping via evolutionary refinement of existing tools.
4. Providing demonstrations of prototypes to the Goddard community for technology transfer and feedback.

Today, the applications initiated by this group are often developed in conjunction with software engineers from other groups. This process results in the transfer of new development technologies to other software engineers.

6. CONCLUSIONS

Software evolution and reuse, given the right environment, works very well. A "life cycle" management structure which allows and encourages evolutionary prototyping techniques is a major requirement for this type of software development. We have demonstrated that the techniques described here can be used to successfully build a satellite scheduling system (EPPS) at nearly half the estimated cost of a traditionally built system. The EPPS was so successful that the development team was awarded a Goddard Productivity Improvement and Quality Enhancement award. Careful management of the toolkit will provide even more payoff for future missions.

7. ACKNOWLEDGEMENTS

The authors would like to thank Patricia Lightfoot at NASA-GSFC/Code 514 and Ellen Stolarik and David Warren at Bendix Field Engineering Corporation for their continual support and many contributions to the IEPS task. The IEPS task is supported by NASA contract #NAS5-27772.

6. REFERENCES

1. McLean, D., Tuchman, A., and Potter, W., "Using C to Build a Satellite Scheduling System: Examples from the Explorer Platform Planning System," *Telematics and Informatics*, Vol. 8, No. 4, pages 297-312, Pargamon Press, 1991.

2. McLean, D., Littlefield, R., and Beyer, D., "An Expert System for Scheduling Requests for Communications Links between TDRSS and ERBS." *Telematics and Informatics,* Vol. 4, No. 4, pages 253-261, Pargamon Press, 1987.

3. McLean, D., Page, B., and Potter, W., "The Explorer Platform Planning System: An Application of a Resource Reasoning Planning Shell," *Proceedings of the First International Symposium on Ground Data Systems for Spacecraft Control,* Darmstadt, Germany, pages 195-200, June 26-29, 1990.

4. Johnson, J., Tuchman, A., McLean, D., Kispert, A., Bogovich, L., Burkhardt, C., Page, B., Littlefield, R., Potter, W., and Ochs, W., "HST Servicing Mission Planning and Replanning Tool," *World Space Congress,* Washington, DC, August 28-September 5, 1992.

5. McLean, D., Page, B., Tuchman, A., Kispert, A., Yen, W., and Potter, W., "Emphasizing Conflict Resolution versus Conflict Avoidance during Schedule Generation," *Expert Systems With Applications,* Vol. 5, pp. 441-446, Pargamon Press, 1992.

6. NASA-GSFC/Code 514, *IEPS Software Toolkit*, Vol. 1-7, August 1992.

7. Arthur, L., *Rapid Evolutionary Development Requirements, Prototyping & Software Creation,* John Wiley & Sons, 1992.